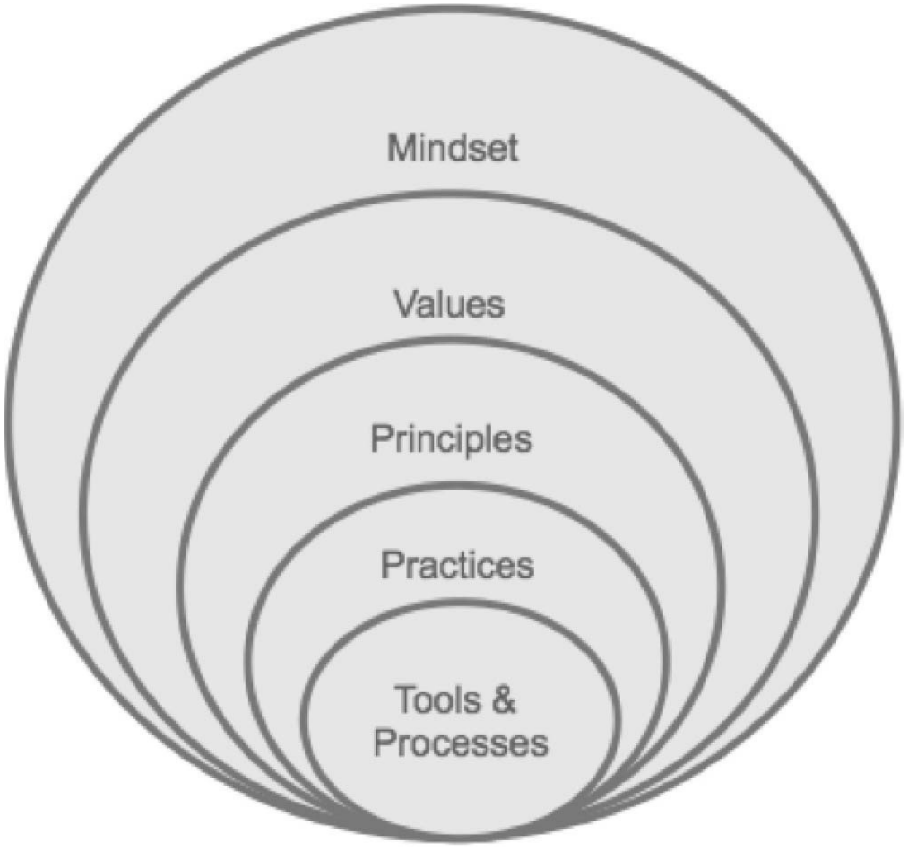
# 2 THE FOUNDATIONS OF AGILE

## 2.1 THE AGILE MINDSET

Figure 2.1 shows the different levels of Agile, from the tools and processes that operate within Agile practices, which in turn operate within Agile principles, up to everything operating within the Agile mindset. An Agile mindset implies that an organisation or person has absorbed Agile to the extent that it becomes part of their 'identity', i.e. their 'business-as-usual' state and the default way they interact with the world.

**Figure 2.1 The Agile mindset wraps around everything**




Agile is not about 'doing' Agile, it is about 'being Agile' and having an Agile mindset, and there are a number of tools, processes, practices and so on that facilitate this. An example could be:

**Tool** An Agile planning tool provides a visual board (see Section 8.7)

**Practice** Visual boards are common practice in most Agile frameworks

**Principle** The practice of using visual boards aligns to the empirical process principle of transparency (see Section 8.7)

**Value** The principle of transparency supports the Agile value of 'responding to change' (makes need for change and the resultant changes transparent)

**Mindset** All of these things are wrapped up in the Agile mindset

It is important to understand that, for Agile to be successful, the right mindset is equally, if not more, important than merely implementing Agile tools, practices or principles – for example, having a visual board does not necessarily mean that a team is Agile.

Experiments have shown that people can strongly influence each other to adopt or reject an Agile mindset (Dweck, 2012). Agile is a journey, not a destination and the best the organisation can hope for is that teams become more Agile by embedding the Agile mindset deeper inside themselves and the organisation. This process is facilitated by applying Agile values, practices, principles and so on.

Table 2.1 lists the characteristics of an Agile mindset and contrasts it with a non-Agile 'fixed' mindset (Rising, n.d.).

**Table 2.1 Fixed and Agile mindsets**

| | Fixed mindset | Agile mindset |
|---|---|---|
| *Ability* | Static, like height | Can grow, like muscle |
| *Goal* | To look good | To learn |
| *Challenge* | Avoid | Embrace |
| *Failure* | Defines identity | Provides information |
| *Effort* | For those with no talent | Path to mastery |
| *Reaction to challenge* | Helplessness | Resilience |

The most important aspect of an Agile mindset is understanding that Agile is neither just a set of rituals that are repeated, nor is it merely based on techniques.

## 2.1.1 Agile is not just a set of rituals

Sometimes, in an Agile transformation, the practices, principles, values and so on are only implemented with partial success. This can create a 'cargo cult' situation (see Box 2.1), where teams may understand what to do, but don't understand why they are doing it. If this happens teams tend to use Agile practices for a short period of time, but over the longer term they fall back into their previous ways of working, simply because they don't understand why they are doing what they are doing.

**BOX 2.1 THE 'CARGO CULT' – AN EXAMPLE OF RITUALS**

Some indigenous islanders on a South Sea island during the Second World War observed soldiers enabling parachute drops from an aircraft on an airfield that had been built on their island. To the islanders, it seemed like the military personnel signalled into the sky, and then food and equipment came from the sky.

In the mid-1970s, a long time after the airstrip had been decommissioned, a team of anthropologists visited the island. What they found was that an entire cult had been built around the idea of signalling into the sky in the hope that food would continue to fall out of it. This is actually a very logical thing to do based on an observation, however the South Sea Islanders had understood what to do but they hadn't understood why they were doing it.

Source: Feynman (1974)

### 2.1.2 Agile is not just techniques

People often prefer to pick out the elements of a framework they can easily understand rather than understanding an entire framework and why it should be implemented. This can lead to people equating the whole of an Agile framework with one or two individual elements of it and, inevitably, it will be difficult or even impossible to realise the whole benefits of implementing Agile. Some Agile frameworks (like Scrum – see Section 14.2) will only work effectively if all the key activities, roles and artefacts of the framework are in place.

So, for example, people might think that just prioritising stories within a backlog (see Section 7.1) is the same as 'being Agile', instead of it being just one part of the whole. Additionally, while prioritisation of stories is indeed a core practice in Agile, it is also used in more traditional non-Agile delivery approaches.

## 2.2 DELIVERY ENVIRONMENTS AND AGILE SUITABILITY

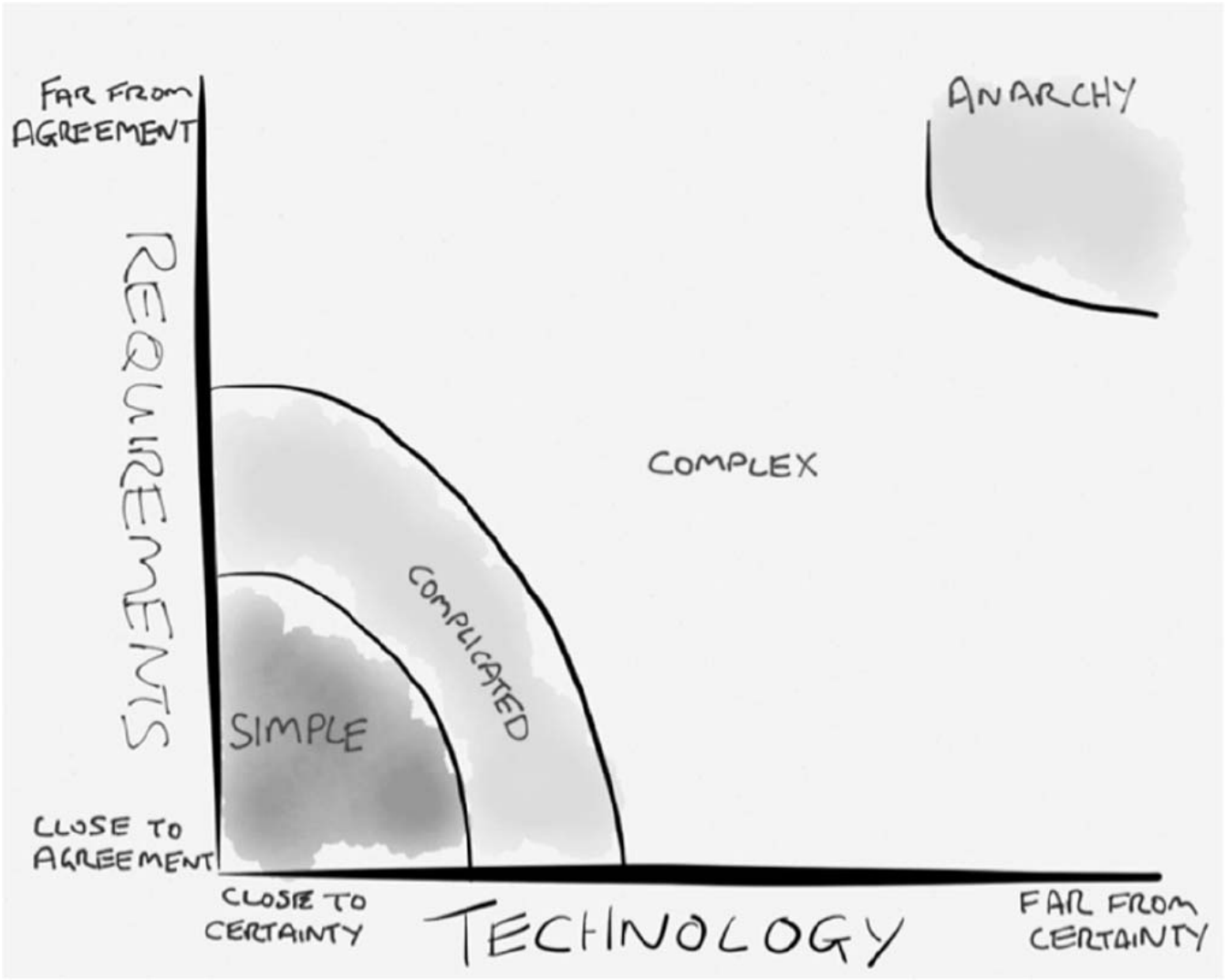
The environment within which delivery will occur should largely drive the delivery and governance framework(s) that will be implemented. For example, in a delivery environment where high variability is likely to be encountered (like IT product development), an Agile framework would be suited; in an environment where variability is likely to be low, a more defined process may be more suited (like 'Waterfall'). See below for more information.

### 2.2.1 Stacey's complexity model

Figure 2.2 illustrates an IT specialisation of a complexity model by Professor Ralph D. Stacey (Stacey, 1996). This model postulates that simple environments – from a perspective of the amount of variation and change they will experience – are those where customer and team have agreed requirements and the technology is close to certainty early within the delivery cycle.

---

**Figure 2.2 Stacey's complexity model**




In a simple environment it is possible to use a defined process, which expects the requirements, technology, design and so on to be largely defined before moving onto the next stage. The Waterfall model (see Section 2.6.3) is an example of a defined process. This means that in some instances what appears to be a very complex project, such as building a bridge, may actually be in the simple area because it may not experience a lot of requirements or technology variation through the delivery lifecycle.

In contrast, in complicated, complex or anarchic environments it is important to use an empirical (learning) process where product increments are developed frequently and consistently in order to obtain feedback and to make sure that the finished product aligns with the evolving business environment. Agile frameworks are empirical processes (see Section 2.6).

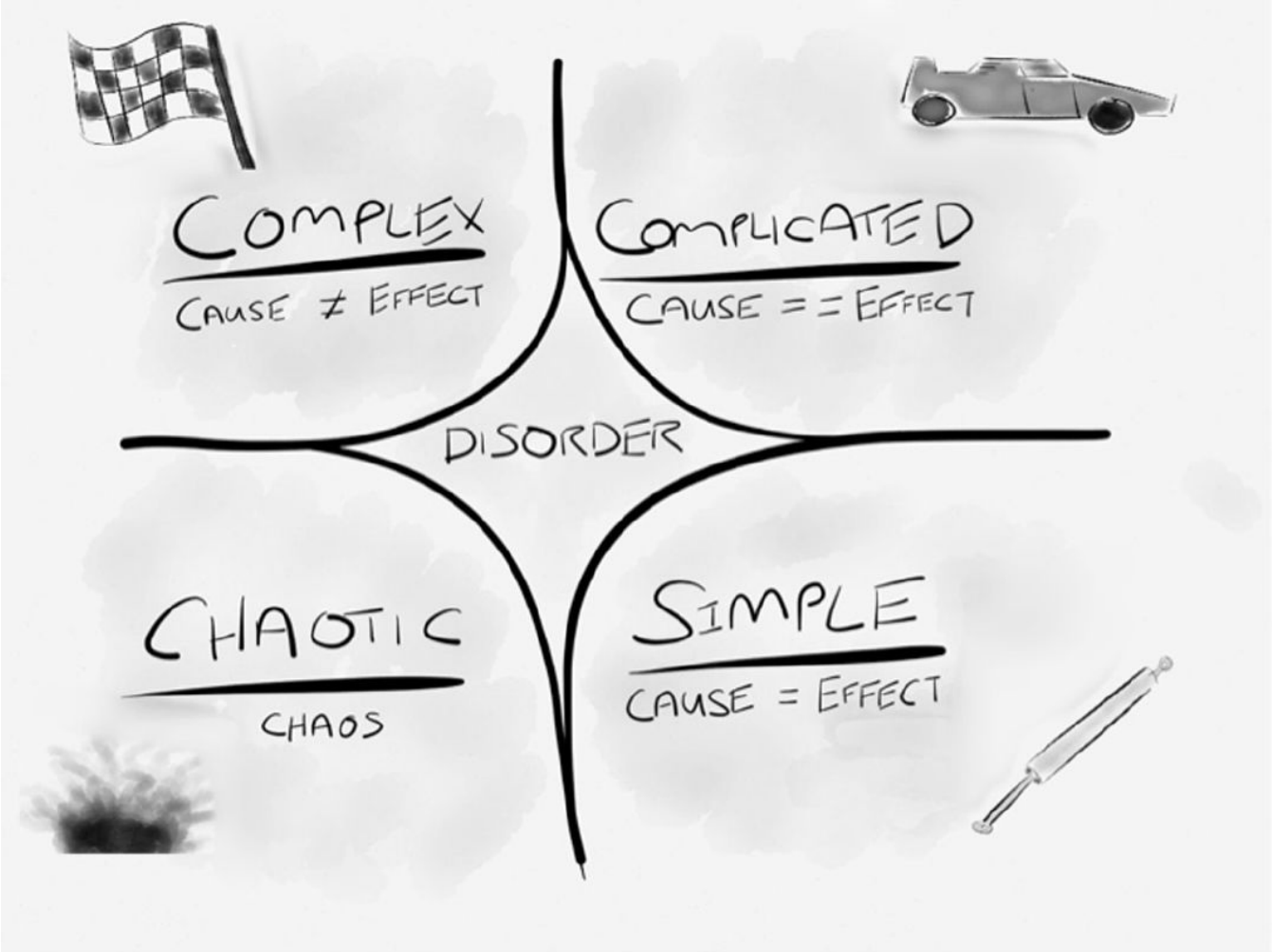
Most modern knowledge-work environments (like IT) are likely to fall into the complicated, complex or anarchic areas because, by their nature, they tend to be innovative and operate in environments where requirements and technology have a high degree of variability through the delivery lifecycle. This is reflected in the Agile frameworks that have been developed for these types of environments: they are empirical and enable teams to run experiments, observe and understand the results, and then adapt processes as appropriate to improve the certainty and quality of delivery.

When trying to understand types of environments, it is important to take into account the amount of innovation that is being sought or considered for a new product or service. As the level of innovation increases, so does the move towards complexity, and a high variability is likely to be present.

### 2.2.2 Cynefin framework

The Cynefin framework (Snowdon and Boone, 2007; Figure 2.3) gives an alternative framework for determining and understanding simple, complicated and complex environments.

**Figure 2.3 Cynefin framework**




Cynefin identifies five domains:

- **Simple (obvious) domain** In this domain the relationship between cause and effect is obvious and therefore it is relatively easy to predict an outcome. In this domain predictive planning works well as everything is pretty well understood. Teams can define up front how best to deliver a product, and they can then create a defined approach and plan. The Waterfall model works well in these types of environments with little variability.
- **Complicated domain** In this domain, the relationship between cause and effect becomes less obvious; however, after a period of analysis it should generally be possible to come up with a defined approach and plan. Such a

plan will normally include contingency to take into account the fact that the analysis may be flawed by a certain amount. Again, the Waterfall model is suitable for this environment as there is an element of definition up front; however, a more empirical process, like Agile, may be more suited.

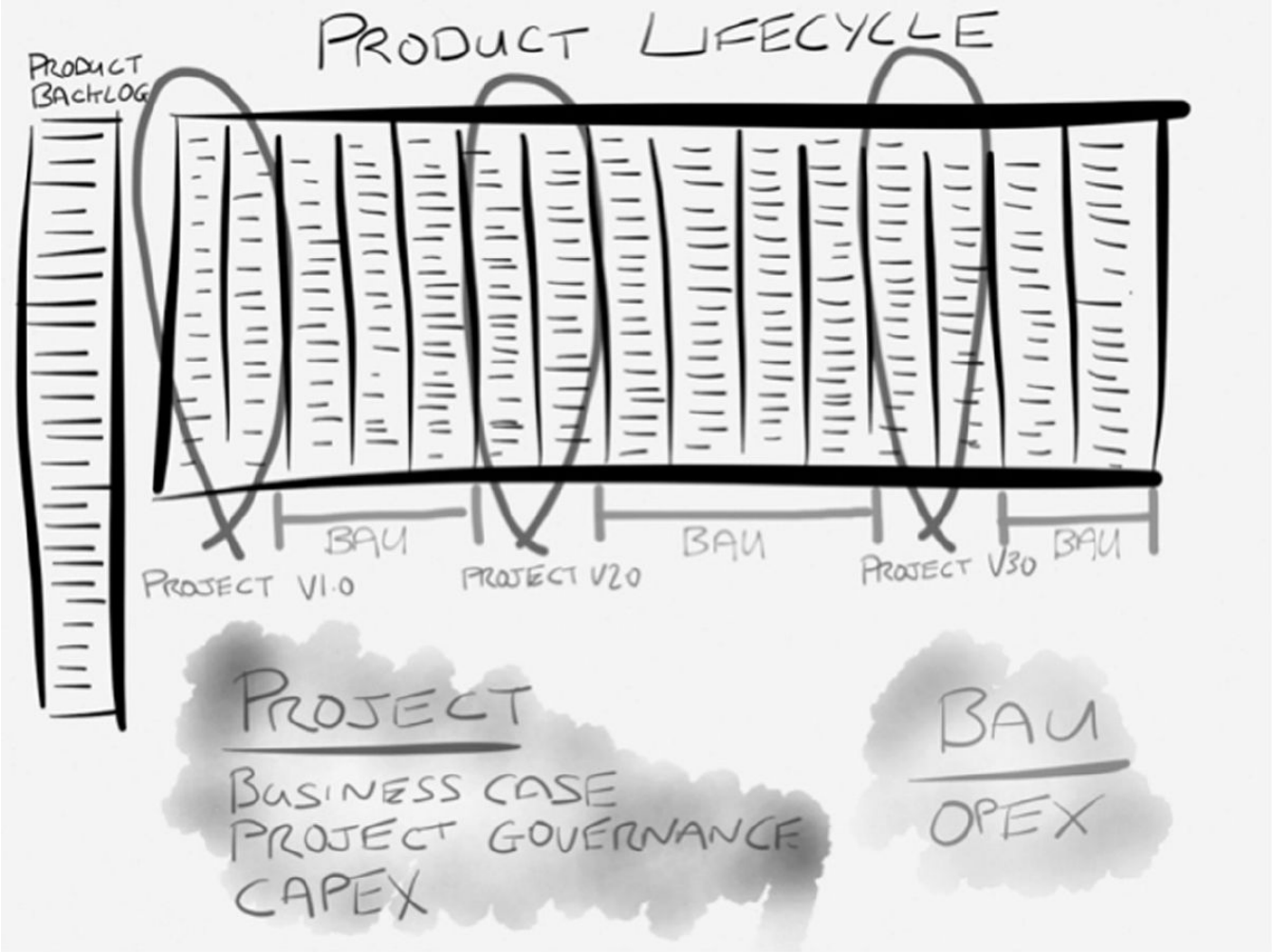
- **Complex domain** In this domain the relationship between cause and effect starts to break down as there tend to be many different factors that drive the effect. While it may be possible to identify retrospectively a relationship between cause and effect, the cause of an effect today may be different to the cause of the same effect tomorrow. Creating a defined up-front approach and plan is not effective within this domain and therefore an Agile way of working is recommended.
- **Chaotic domain** In this domain, there is no recognisable relationship between cause and effect at all, making it impossible to define an approach up front or to plan at all. Instead, teams must perform experiments (e.g. prototyping, modelling) with the aim to move into one of the other less chaotic domains. An Agile approach can work in this domain, for example Kanban (see Section 14.5), which does not require up-front plans.
- **Disorder** Being in this environment means that it is impossible to determine which domain definition applies. This is the most risky domain as teams tend to fall into their default way of working, which may prove unsuitable for what they are trying to achieve.

During a product's development and evolution there may be elements of delivery spread across all the Cynefin domains at the same time. There may be aspects of a large system that are simple, while others may be in the complicated domain; and there could also be areas where innovation is necessary and which require a move towards the complex or even towards the chaotic domain.

## 2.3 THE LIFECYCLE OF PRODUCT DEVELOPMENT

This section will give an overview of the lifecycle of a product from initial commissioning to final decommissioning and discuss how project and business-as-usual (BAU) delivery frameworks may be used throughout the lifecycle of the product (see Figure 2.4).

### Figure 2.4 The lifecycle of product development




The full lifetime of any product spans from the commissioning of the product to its decommissioning. Depending on the business that is implementing the product this could be anything from a few weeks or months (a short marketing campaign), to many decades (a bank's accounting system).

What drives product development in an Agile approach is an evolving backlog of stories (see Section 7.1) that the customer wants and that are placed in an order by the customer. The backlog exists throughout the whole product lifetime, from commissioning to decommissioning. The customer evolves the backlog with the stakeholders and the team. Typically the stakeholders support the customer to define coarse-grained stories in the backlog and the team supports the customer to define fine-grained stories in the backlog.

There are generally three styles of Agile delivery:

- Defined product via a BAU delivery – This style assumes that the customer wants a high-level definition of the product and will assign funding and resources over a time frame to deliver that product, which may or may not be defined in a business case. The team and customer will evolve the product within the available timescale and resources, prioritising what will be delivered.
- Defined product via a project delivery – This style assumes that the customer wants a high-level definition of the product and will assign funding and resources over a time frame; however, the complexity of delivery (or

preference of the customer) means project governance will be in place. Projects must have a business case. The team and customer will evolve the product within the available timescale and resources governed within a project framework to monitor delivery against a business case.

- Undefined product via a BAU delivery – This style of delivery assumes that the customer doesn't have an end point definition of a product or specific timescale or cost in mind up front. The customer and team will define what increment of the product they want in the next iteration/sprint and then create that increment. Once that increment has been delivered the next delivery increment is planned and delivered in the next iteration/sprint. The delivery continues until funding runs out or it doesn't make sense to evolve the product any further. This creates an environment where delivery can be extremely flexible and emergent, which is excellent for highly variable delivery environments. Arguably this style of delivery is the nearest to what some people may consider to be 'pure Agile'.

Whichever style of Agile is being applied (and it may be all of them across different teams) a key focus of Agile is enabling effective product flow, which will therefore enable continuous delivery of value to the customer. Effective product flow is enabled by many factors, one of which is batch size. Projects and releases can lead to large batch sizes which in turn can lead to, amongst other problems, errors and low productivity (see Section 10.1.1). Therefore, when implementing Agile the delivery sprints/iterations (batch sizes) should be as small as possible (2 to 4 weeks is typical), and shippable product should be produced at the end of each sprint/iteration. Grouping iterations/sprints into releases or projects can lead to large batch sizes, which is sub-optimal, however there may be strong business reasons (like the customer only wanting quarterly deliveries) why this must happen. Agile people talk about 'potentially shippable increments' to deal with this issue.

There is sometimes confusion about whether project management frameworks can or should be wrapped around Agile frameworks. This is often due to a fundamental misunderstanding of what project management frameworks are, and concerns that the frameworks will be delivered in a 'command and control' style that will stifle agility. However, all project management frameworks – including PRINCE2 (Projects IN Controlled Environments (Axelos, 2014)) and PMBoK (Project Management Body of Knowledge (PMI)) – are designed to wrap around many other frameworks, including Agile frameworks; however, they must be customised.

As an example, PRINCE2 consists of seven guiding principles. The seventh principle guides users to customise their delivery approach depending on the environment in which it will be used:

> PRINCE2 is tailored to suit the project's environment, size, complexity, importance, capability and risk.
>
> (Axelos, 2014)

Therefore when using PRINCE2 with Agile, PRINCE2 should be customised for

the Agile environment. So, for example, the ‘Manage Stage Boundary’ milestones in PRINCE2, which are aligned in a Waterfall delivery to end of analysis, end of design, end of build and so on, will, when integrated with an Agile framework, actually relate to delivery of outcomes at release boundaries, and not to Waterfall stage gates and documents. This means what will be assessed is the delivery of outcomes, rather than the delivery of products associated with the Waterfall stages.

If a project management framework is required in conjunction with an Agile framework that does not provide its own project governance, there is a standard Agile framework that has been designed to wrap around other Agile frameworks. This is called ‘AgilePM’ (Agile Project Management – see Section 14.4) which is part of the ‘DSDM Agile Project Framework’ (see Section 14.3).

## 2.4 THE ‘IRON TRIANGLE’

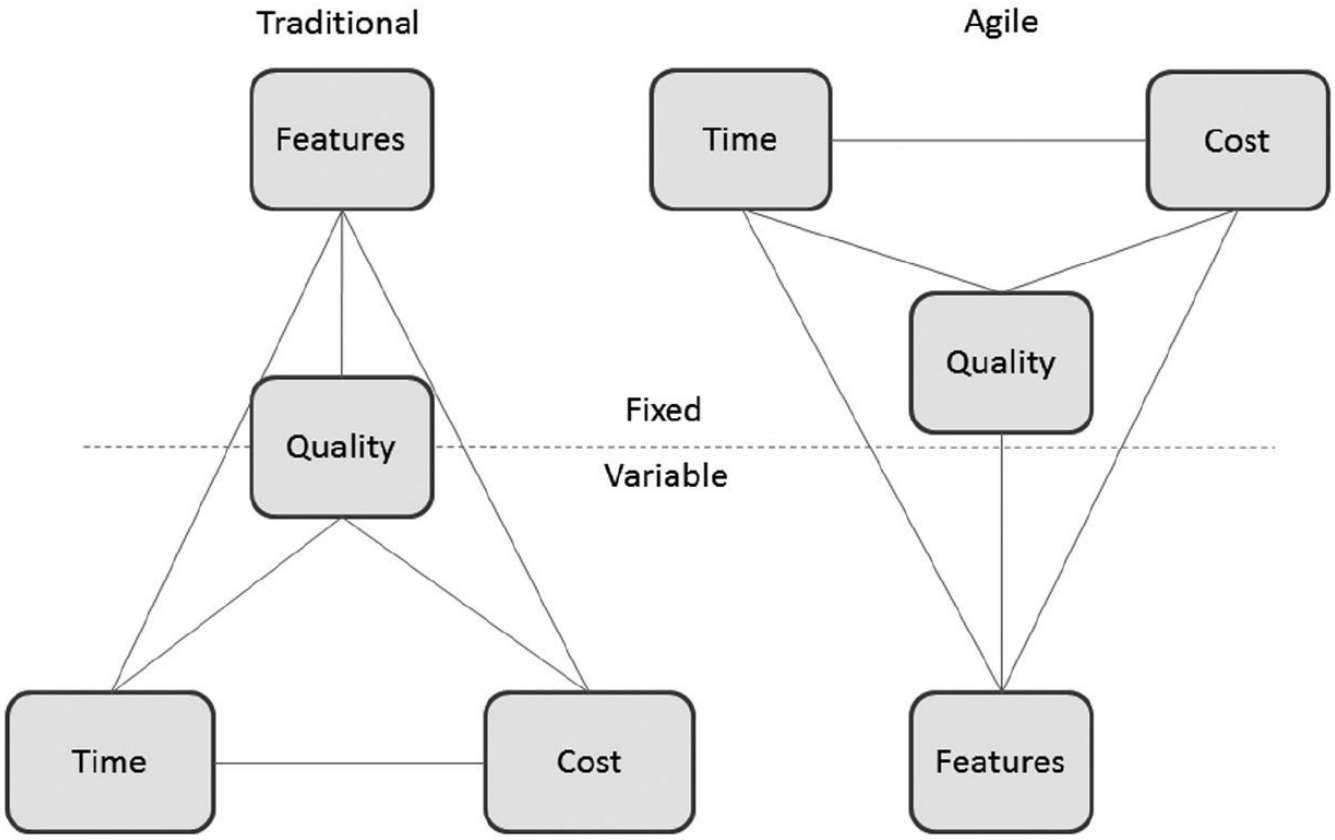
The ‘Iron Triangle’ (see Figure 2.5) is a means to show the constraints that need to be kept in mind when developing products as changes in one constraint affect the others. The traditional primary constraints are features (requirements), time and cost, but we have added technical quality as a fourth constraint. In a delivery, teams and customers have to choose which particular constraint(s) they wish to fix, if indeed they fix any of them, and which ones they want to keep as dynamic elements. This section will discuss broadly how Waterfall and Agile approaches fix and vary the constraints of the Iron Triangle.

### 2.4.1 The Waterfall implementation of the Iron Triangle

The Iron Triangle on the left in Figure 2.5 above shows a traditional Waterfall approach (see Section 2.6.3), where all features are defined (and possibly fixed) up front and a project is planned up front to deliver all of those predefined features. Therefore in the Waterfall Iron Triangle the features are largely fixed and become part of an actual or implied contract between the customer and the team. The assumption is that quality, time and cost will be the variable elements. This approach works well for deliveries where the feature requirements and/or technology are simple and unlikely to change (the simple area in Stacey’s model – see Section 2.2.1).

---

**Figure 2.5 The ‘Iron Triangle’**




However, this approach can cause problems if operating in an environment that experiences variability. For example, if the features are set pre-delivery, they become the fixed point on the triangle. If the customer then needs to change what they want or if the up-front analysis and design prove to be incorrect, potentially expensive and invasive change control has to be introduced, meaning that time and cost may grow significantly.

Most organisations need IT solutions to be delivered on time, simply because IT is such a significant enabler of competitive business advantage. Therefore when things start to vary, expand or go wrong, the business needs to find a way to bring the project back on schedule. In a traditional Waterfall model, where the features are fixed there is a risk that this is done by simply adding more people to the development team. This, in turn, introduces the potential for increased risk, cost and complexity.

There have been many white papers and books written about the risks of simply throwing people at a project that is experiencing delivery problems. One of the fundamental problems of this approach is that the people who are already in the project team will need to spend significant unplanned time and effort on bringing people who are new to the team up to speed. This will typically cause time to slip further.

Probably the most famous book that discusses this is *The Mythical Man-Month* (Brooks, 1995), which asserts that adding manpower to a late software project just makes it even later (Brooks' law).

It describes the sequence of problematic events as follows:

- A time delay leads to more people, and the associated costs, to be added to the

project.

- Bringing those people up to speed causes further time delay (because of the 'learning curve'), so leading to more people and more associated costs being added to the project to bring it back on schedule.
- This causes further time delay, so the team add more people and the associated…and so on until a very costly project failure.

Another significant risk of using a Waterfall approach in a dynamic environment is the possible effect on quality. If the features change, quality might be squeezed or even ignored in an attempt to meet the all-important delivery date. For example testing might be shortened and de-prioritised, architectures and designs may be compromised, and possibly detrimental shortcuts may be allowed to 'make the software work' in the immediate term, eventually leading to technical debt (see Section 10.4). This can be exacerbated by the difficulty of changing detailed specifications that have been created pre-delivery.

## 2.4.2 The Agile implementation of the Iron Triangle

On the right of Figure 2.5 the triangle is turned on its head; this is the paradigm that Agile creates. Rather than creating detailed feature specifications and delivery plans up front, Agile fixes the time, as well as cost and quality, which enables the features to be reviewed and evolved to stay in line with the dynamic needs of contemporary businesses.

### *2.4.2.1 Time-boxing*

This concept is called 'time-boxing'. Time-boxing divides the delivery of increments of the product into short, manageable time periods (called sprints or iterations) of days or weeks, and varies, based on priority, the functionality to be delivered within these time-boxes. While quality can still be changed, the focus is always on understanding the risk of adding technical debt by changing the quality.

A project is a time-box, a release is a shorter time-box, and an iteration/sprint is an even shorter time-box, and all of these can be controlled effectively by Agile. The goal related to the iteration/sprint time-box is a commitment, whereas the goals related to project and release time-boxes are baselines, which can be changed.

Sceptics sometimes wrongly criticise Agile projects for not defining what is going to be delivered up front. They think that an Agile project delivers 'something' within a time frame and that the development team 'make it up' as they go along – these are incorrect assumptions. It may be true that in a highly variable/chaotic environment the only way to deliver something is on an increment-by-increment basis within very short planning horizons of iterations/sprints. However, typically Agile incorporates a short and effective up-front period of time for set-up work as and where required – for example, in Scrum (Section 14.2) there is a 'Sprint Zero'; in eXtreme Programming (see Section 14.1), there is an 'Iteration Zero'; and in Agile Project Management and DSDM (see Section 14.3) there are 'Feasibility' and 'Foundation' stages.

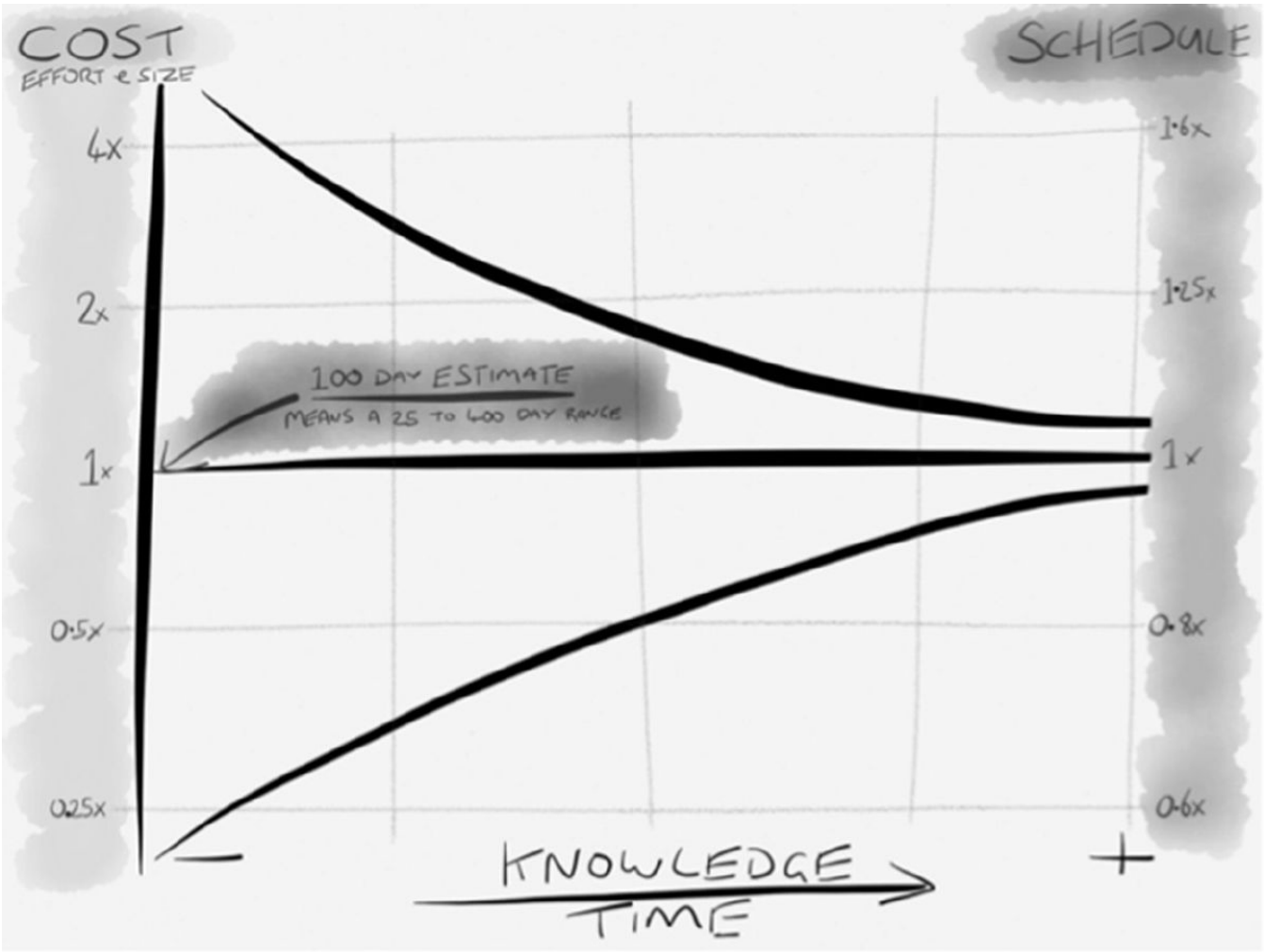
Set-up normally requires a couple of weeks in which a high-level understanding of the vision of the project is achieved, together with a high-level prototype and a high-level technical architecture. These need to be short enough to be accurate and simple enough to be changed effectively and without huge overspend.

## 2.5 WORKING WITH UNCERTAINTY AND VOLATILITY

Early in an Agile delivery little is generally known about the work needed. At this stage, any estimates have higher uncertainty and thus are often provided using a range or with contingencies and the associated costs.

As the delivery progresses and knowledge increases, the level of variation in estimates is reduced to the point when more and more accurate forecasts can be provided. Boehm's Cone of Uncertainty (Boehm, 1981) (see Figure 2.6) explains the relationship between the level of knowledge and the amount of uncertainty that should be expected in providing any estimate based on effort, cost and/or time.

**Figure 2.6 Boehm's Cone of Uncertainty**




The traditional Waterfall model is based in part on the belief that doing significant amounts of analysis and design will create a deep understanding and knowledge of the system. However, this fails to take into account the inherent uncertainty and variability in knowledge-based work such as software delivery.

Accurate knowledge can only be reasonably acquired by interacting with a tangible product as it is delivered. Therefore, the more opportunities the team, customers

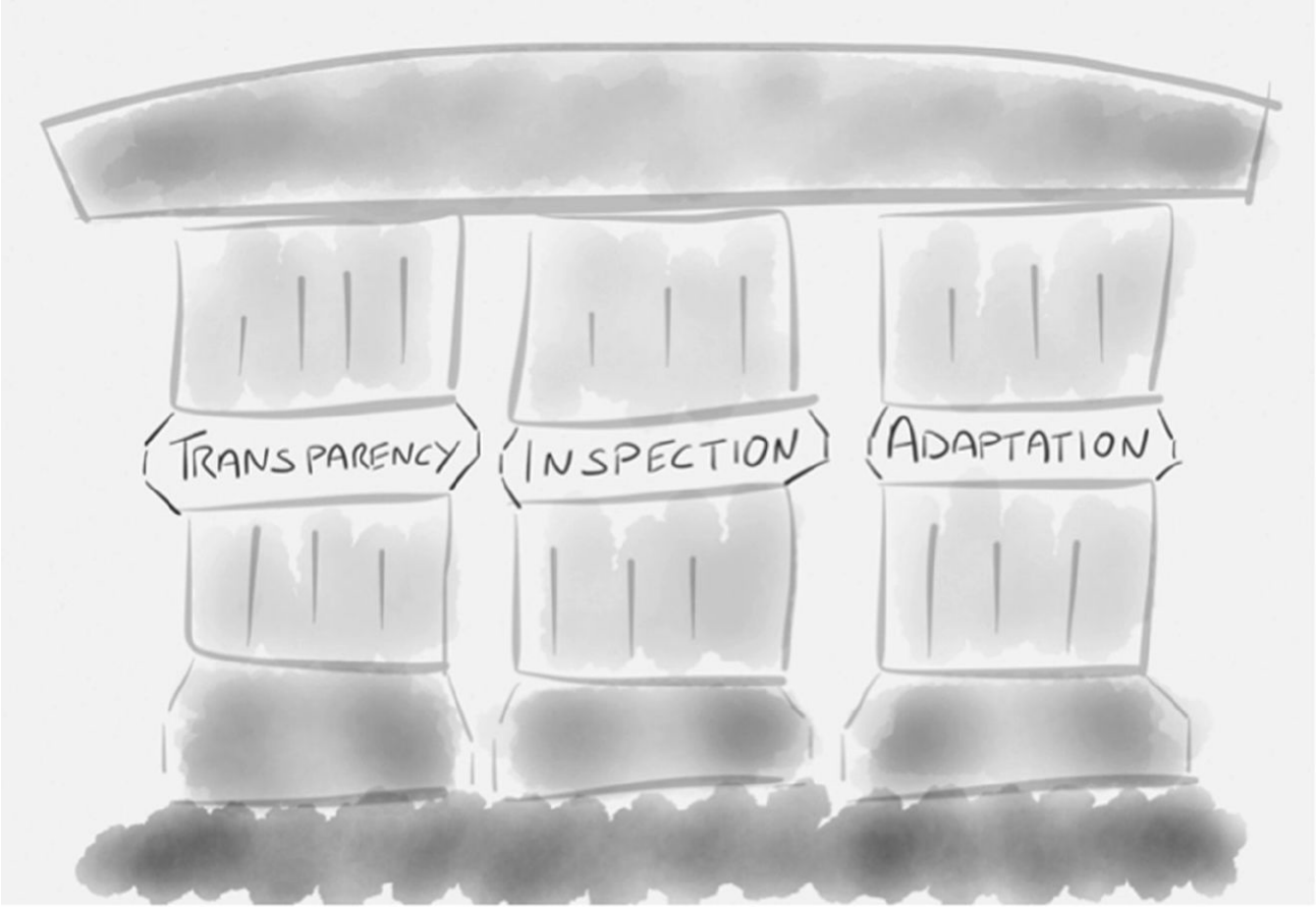
and stakeholders have to experience the product collaboratively, the more opportunity they have to discuss and agree the evolving product requirements and technology needs, and therefore the more accurate their evolving estimates will be.

## 2.6 EMPIRICAL AND DEFINED PROCESSES

### 2.6.1 The three pillars of the empirical process

Agile product development follows an empirical process (a learning process). The three pillars of empirical processes are (see Figure 2.7):

**Figure 2.7 Three pillars of empirical processes**




- Inspection – inspect the product being created and how it is being created
- Adaptation – adapt the product being created or the creation process if required
- Transparency – ensure everyone can easily see what is happening

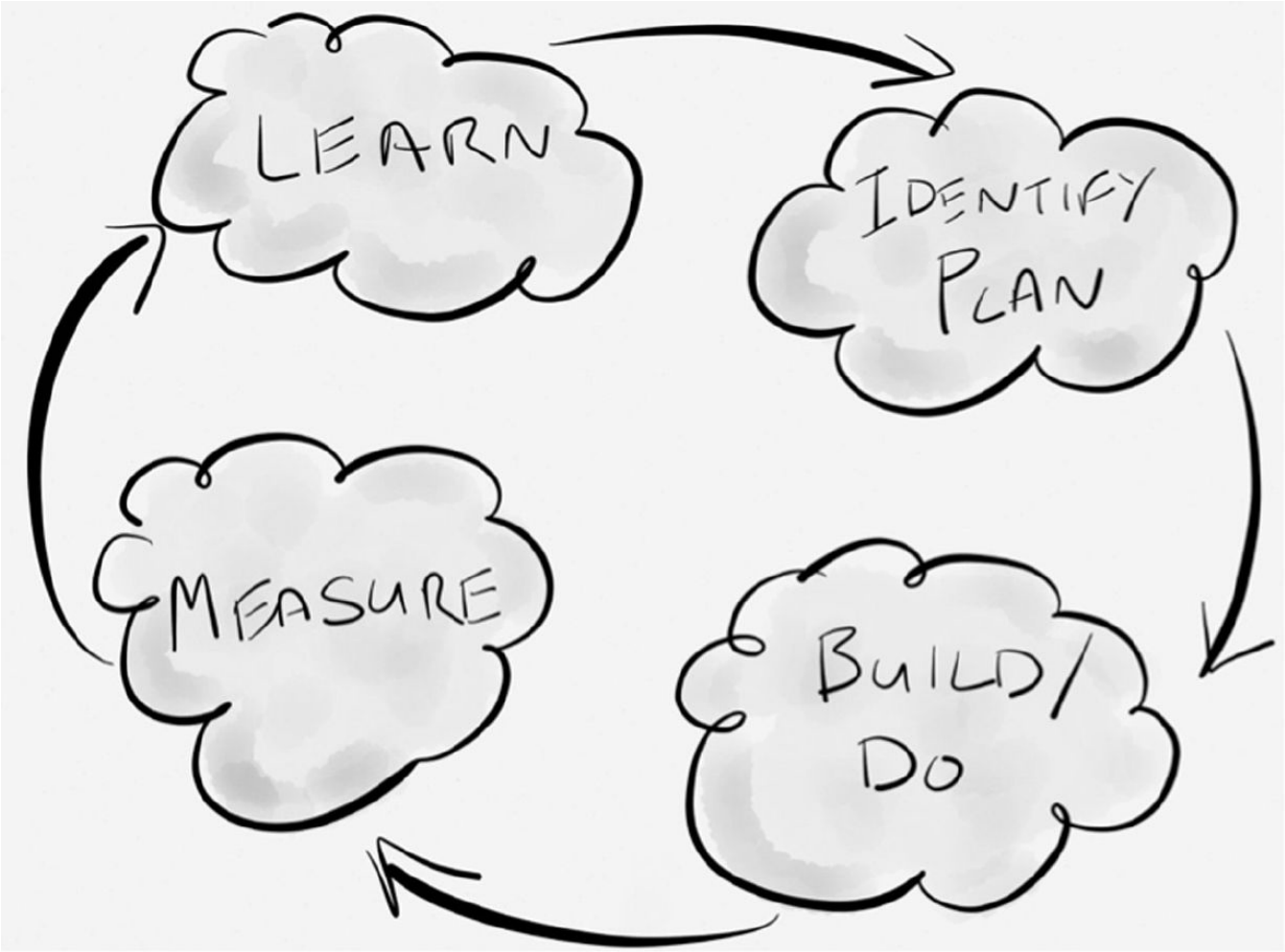
People use empirical processes in their daily lives. It is the way people learn most effectively, i.e. learning through experience and from mistakes (Bryner, 2007).

In an empirical process the team inspect the environment in which they are delivering (see Section 2.2), and then adapt what they are doing and how they are doing it to suit the environment. If the team are going to ‘inspect and adapt’ then they need to make sure that the adaptation is transparent to all stakeholders (see Section 6.4).

### 2.6.2 Empirical processes

Empirical processes (see Figure 2.8) incorporate repeated inspection and adaptation of a product to ensure the right product is delivered in the right way. This is especially important in environments that experience high variability (see Section 2.2) and are therefore most suited to Agile working.

## Figure 2.8 Empirical change model




There are many Agile frameworks (see Chapter 14), however, they all have a similar underlying structure in that they identify something that needs to be done, do it, measure and review the results, and then inspect and adapt based on that knowledge. Some examples of empirical models are as follows:

**PDCA** Plan, Do, Check, Act – Edward Deming (Deming, n.d.).

**POOGI** Process of On-Going Improvement – Theory of Constraints (Goldratt and Cox, 1984).

**OODA** Observe, Orient, Decide, Act – John Boyd (Boyd, n.d.).

**BML** Build, Measure, Learn – Lean Start-up (Ries, 2011).

**DMAIC** Define, Measure, Analyse, Improve, Control (Six Sigma, 2006).

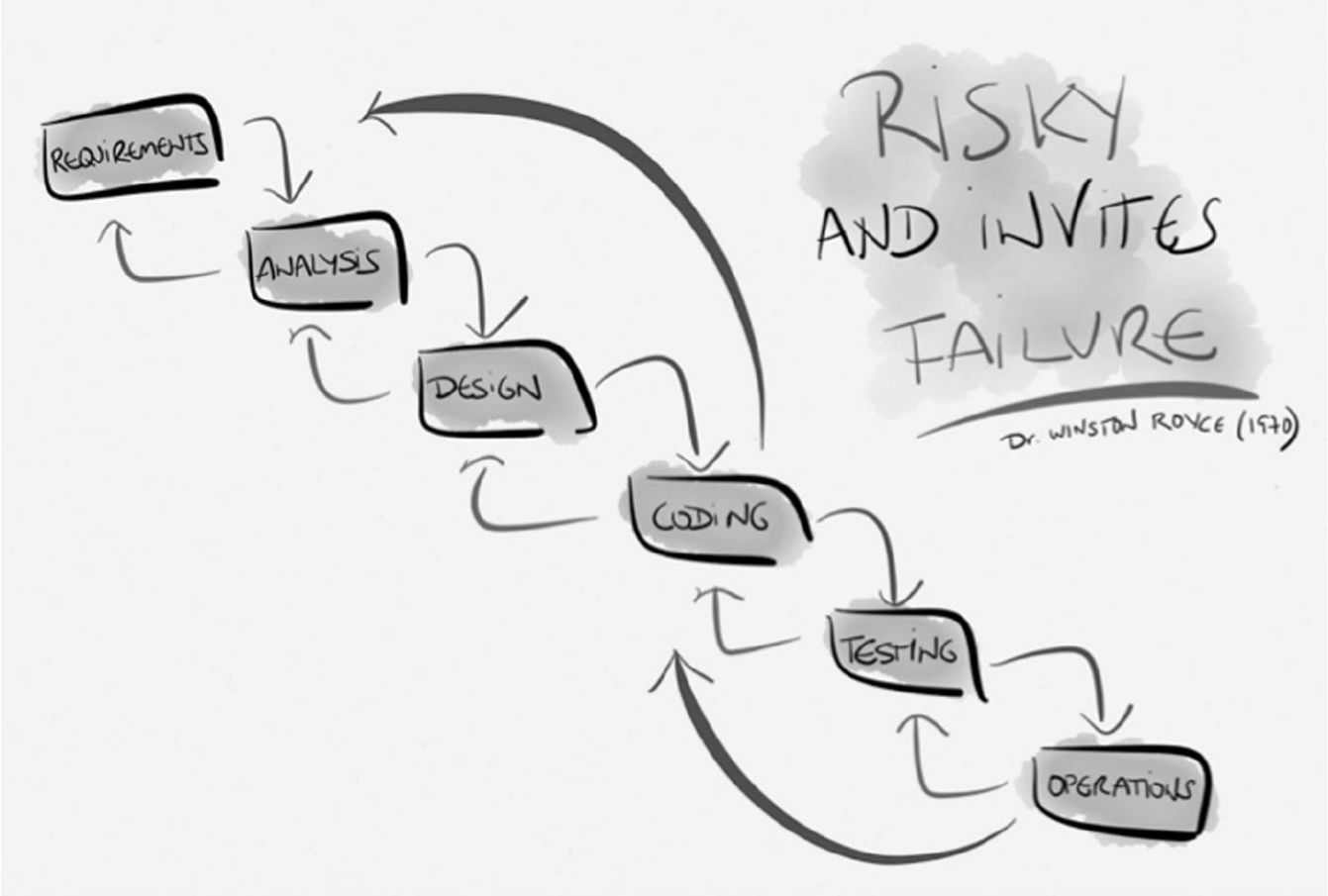
**TAC** Thought, Action, Conversation – DSDM Agile Project Framework (DSDM Consortium, 2014b).

**Kaizen** A Japanese word which means ‘good change’, used to describe a philosophy of continuous improvement (Liker, 2004).

### 2.6.3 Defined processes

One of the best-known defined processes is the Waterfall approach (see Figure 2.9). The Waterfall approach was originally implemented in the civil engineering environment. In these environments the cost of change is typically prohibitive and therefore it makes great sense to define exactly what is required (requirements and analysis), design exactly what is required (design), build it (build), test it works and meets requirements (test), and then make it operational.

**Figure 2.9 The Waterfall approach**




The Waterfall approach was also one of the first formal delivery approaches utilised by the IT industry. It was first cited as an example of how software **could** be delivered by Dr Winston Royce in an article titled *Managing the development of large software systems* in 1970 (Royce, 1987[1970]), although Royce didn’t use the term Waterfall. He also stated, ‘I believe in this concept, but the implementation described above is risky and invites failure’ (Royce, 1987[1970]). Many IT people read Royce’s paper, saw the Waterfall diagram and implemented it, but missed Royce’s warning about using it as described in the diagram.

The arguments for the Waterfall style of delivery are as follows:

- Effort spent analysing and designing up front will lead to greater understanding and also lead to a solution that is less likely to contain errors.

- Ensuring that each stage is rigorously completed will ensure that problems will not occur later in the delivery when they are very costly to fix.
- Communication is via documentation, which ensures that if anyone leaves there will not be a significant impact on the delivery.
- Waterfall implements rigour and predictability via detailed delivery stages that are signed off as milestones.

These arguments are all very robust in a simple domain (see Section 2.2.2). However, in environments like IT where the cost of change is not prohibitive and variability is high, the Waterfall approach does not work effectively due to it not being designed for these environments.